

Korespondenční Seminář z Programování

ZAČÁTEČNICKÁ KATEGORIE

29. ročník

KSP-Z

říjen 2016

Doufáme, že jste si užili prázdniny, a jsme rádi, že jste na jejich program zavřadili i řešení úloh, které jsme na vás přičtystali. V tomto letáku najdete jejich autorská řešení.

Jako již tradičně, iště týden po zveřejnění řešení necháváme otevřené Odezdvátátko. Pokud jste tedy řešení nějaké z praktických úloh zatím nevymysleli, můžete si ho alespoň zkusit naprogramovat a získat až třetího bodu.

Jako vždy, pokud se vám cokoli mezdá nebo máte nějaký dotaz, neváhejte se ozvat na našem fóru.



Řešení první série začátečnické kategorie 29. ročníku KSP

29-Z1-1 Kevinova želva

Pozice želvy je dána dvěma souřadnicemi, proto nám stačí dvě proměnné, do kterých si budeme průběžně ukládat polohu želvy.

Ze vstupní budeme číst písmenka směrů a podle nich vždy upravíme souřadnice, které tento pohyb změnil. Mohou tedy nastat čtyři případy:

Pokud jde želva na sever (S), tak se pohybuje kladným směrem a její souřadnice y se tedy zvětší o 1, zatímco x se nezmění.

Naopak pokud půjde na jih (J), jde opačným směrem a y se zmenší o 1.

Ve směru na východ (V) se pro změnu zvětší pouze souřadnice x o 1.

A pokud želva míří na západ (Z), její souřadnice x se zmenší o 1.

Program (Python 3):

```
http://ksp.mff.cuni.cz/viz/29-Z1-1.py
```

Pro tyto účely má velká část programovacích jazyků konstrukci, která se jmenuje *svička* a umožňuje smažit větvení programu podle proměnné, která může nabývat několika různých hodnot.

Program (C):

```
http://ksp.mff.cuni.cz/viz/29-Z1-1.c
```

Také můžeme využít slovníku, který uchovává změnu souřadnic pro každý možný směr. To je dobrý přístup, zvláště kdybychom měli ještě více směrů, kterými se želva může vydat.

Program (Python 3):

```
http://ksp.mff.cuni.cz/viz/29-Z1-1-dict.py
```

Jirka Sejkora & Misa Šolová

29-Z1-2 Sáňky panámků

Naším úkolem bylo vypsat čísla z intervalu, přičemž násobky tří jsme měli nahradit křížkem, násobky pěti kolečkem a násobky obou vynechat.

Základní myšlenka je jednoduchá: Projdeme v cyklu všechny čísla z intervalu a u každého zkontrolujeme, jestli je násobkem 3 nebo 5, a podle toho s ním naložíme. Víc nembylo třeba vymýšlet, zato bylo potřeba tuhle myšlenku naprogramovat. Co nás tedy může zaskočit?

Hranice intervalu – první číslo se bere včetně, druhé bez. To ovšem můžeme jednoduše vykonat ze zadání. Takové



pojetí intervalů je navíc v programování obvyklé, mimo jiné proto, že se pak přírůdek přirovnáve zapisuje (např. v Pythonu) jako `for i in range(od, do)`.

Testování dělitelnosti: Zde se určitě hodí znát operátor modulo, který je ve většině programovacích jazyků a obvykle se zapisuje procentem, $\%$. Modulo nám vrátí zbytek po celočíselném dělení. Jestli je číslo x dělitelné třeba 3, tak jednoduše otestujeme jako `if x % 3 == 0`.

Dělitelnost obojím. S čísly, která jsou násobky jak 3, tak 5, musíme být opatrní. Asi nejjednodušší přístup je na začátku zkontrolovat, jestli je číslo dělitelné obojím, a pokud ano, přeskóčit ho (v Pythonu bud použijm `continue`, nebo umístěním zbytku těla cyklu do `else` větve).

Pokud předpokládáme, že modulo funguje v konstantním čase, pro každé číslo z intervalu provedeme maximálně konstantní počet kontrol a maximálně jedno vypsaní, tedy časová složitost je $O(N)$, kde N je velikost intervalu.

Čísla, resp. nahrazující znaky, můžeme rovnou vypisovat na výstup, paměťová složitost je tak konstantní, $O(1)$.

Pro zajímavost ještě dodáme, že naše pohádka skrývá jinak velmi známou úlohu, obvykle označovanou jako *Fizz Buzz*. Občas ji někdo používá při vstupním pohovoru na programovací profese.

Program (C):

```
http://ksp.mff.cuni.cz/viz/29-Z1-2.c
```

Program (Python):

```
http://ksp.mff.cuni.cz/viz/29-Z1-2.py
```

Karoly Buršková

29-Z1-3 Petrova statistika

Naším úkolem bylo graficky znázornit, kolikrát želva dostala jaký počet panámků. Podobným grafem se většinou říká *histogram*. Řešení není vůbec těžké, stačí ovládat práci s poli.

Počítáme si jedno velké pole, říkáme mu třeba *četnost*. V poli na indexu i budeme uchovávat počet dní, ve kterých dostala želva i panámků. Pokud to váš jazyk neudělá sám, je potřeba ho na začátku naplnit nulami.

Jak bude pole velké? Ze zadání víme, že počet panámků nepřesáhne 1 000 000. Jeden milion čísel se nám do paměti vejde i pokud zkonštruje řádky KSP na telefonu.

Pak už jen proložíme postřipné počty panámků a za každé i přičteme k *četnost*[i] jedničku. Tím máme připravena data, která chceme vypsat.



matfyz

KSP pro vás připravují studenti Matematicko-fyzikální fakulty Univerzity Karlovy.

Webové stránky:

<https://ksp.mff.cuni.cz/>

E-mail:

ksp@mff.cuni.cz

Diskusní fórum:

<https://ksp.mff.cuni.cz/forum/>

Chcete-li s námi komunikovat bezpečně, můžete si ověřit náš HTTPS certifikát – jeho SHA1 fingerprint je: E9:DB:EE:C6:62:BC:14:DE:09:E4:E8:97:DC:36:0E:87:B3:50:B0:01.

Zadáni po nás ovšem chtělo, abychom ukázali jen relevantní část histogramu. Proto si ještě spočítáme minimum a maximum ve všech i , která jsou počtali. To můžeme dělat průběžně, nebo si čísla načíst všechna a projít je ještě dvakrát.

Pokud by se nám nelíbilo, že si vytváříme zbytečně velké pole, můžeme počítání minima a maxima provést ještě před počítáním četností. Potom už nám nic nebrání pořádit si pole jen pro hodnoty mezi extrémny.

Leptším řešením je však použít slouvník. Co to je se můžete dočíst v našich kuchárkách, ale například v Pythonu jej použijete velice snadno. S trochu odlehčením stačí napsat `hnanet` závorky složeným, případně použít kolekt `defaultdict`, jako to dělá úkázkový zdrojový kód

Program (Python 3):

`http://ksp.mff.cuni.cz/viz/29-21-3.py`

Program (C):

`http://ksp.mff.cuni.cz/viz/29-21-3.c`

Ondra Hlavatý

29-21-4 Zuzčin výlet

Zuzka chce vymyslet, jaká posloupnost kterých skluzavek je třeba k tomu, aby jích zkusil za jednu cestu dohl a nejvíce. K takovému zkomunikání je dobré použít nějaký systém, aby na základem možnost nezapomněla, ale také aby jí takové hledání netrvalo zbytečně dlouho.

Zuzku napadlo, že bude postupovat zprava doleva – to znamená, že se podívá v mapce na skluzavku, která vede z horní plošiny co nejvíce vpravo. Poté vybere další, pro kterou platí dvě podmínky: aby na tu předchozí navazovala a aby vedla zase co nejvíce vpravo. Takhle postupuje dál a dál, dokud nebude žádná další skluzavka nebo tobogán, který by navazoval. Teď si Zuzka poznamená, kolik různých skluzavek by cestou dolů vyzkoušela a podívá se po jiné variantě.

Pro zachování systému se vrátí o jednu skluzavku nahoru, odečte jedna od poznamenaného čísla a bude pokračovat zase dolů – ale jinudy. Nyní zvolí možnost, která nebude ta nejpravější, ale o jednu více vlevo, a opět postupuje směrem dolů a vpravo. Když už nebude moci nikam dál, zase si poznamená, kolik celkem skluzavek by vyzkoušela v této variantě od horní plošiny až sem dolů. Občas se ale bude muset vrátit dokonce více než o jednu skluzavku, aby skutečně vyzkoušela všechny možnosti oshora dolů.

Takhle Zuzka postupuje, až dokud neprojde variantu, která je úplně na opačné straně – co nejvíce vlevo. Teď má poznamenanou několik údajů, které říkájí, kolik nejvíce skluzavek a tobogánů lze vyzkoušet v různých variantách. Nakonec z nich stačí vybrat největší číslo. Takovému postupu procházení skluzavek se říká procházení grafu do hloubky a více si o něm můžete přečíst v našich programátorských kuchárkách.¹

Na závěr ještě jedno vylepšení předchozího programu. Aby si Zuzka nemusela poznaménávat zbytečně mnoho čísel, stačí, když si vždy bude pamatovat jen dosavadní maximum a když najde cestu s více skluzavkami, zapamatuje si to.

¹ `http://ksp.mff.cuni.cz/viz/kucharky/grafy`

² `http://ksp.mff.cuni.cz/viz/kucharky/trideni`

³ `http://ksp.mff.cuni.cz/viz/kucharky/hledani-v-textu`

Program (Python 3):

`http://ksp.mff.cuni.cz/viz/29-21-4.py`

Zuzka Družalová

29-21-5 Dva seznamy

Kevin a Sara si po chvíli rozhodli všimni, že vždy když je zajímavá, jestli je daný kamarád na některém seznamu, musí ho celý přečíst. Sárku proto napadlo, že by bylo nejlepší si seznamy seřadit.

Pro jednoduchost si můžeme představit, že kamarádi nemají jména ale čísla a budeme třdit ta. Pokud bychom pracovali se jmeny, musíme si třídící algoritmus upravit pro práci s řetězci. Jeho složitost by pak byla přibližně stejná jako nejlepšího řetězce, což jsme ovšem v zadání omežili na konstantu 42, takže nás nezajímá.

Třdit je možné kromou různých algoritmů, o některých pojednává naše kuchárka o třídění.² O té slyšel už i Kevin a po prostudování textu se rozhodl pro MergeSort.

Když už máme seznamy seříděné, bude nejlepší postupovat tak, jak radí intuice: Kevin i Sara si budou prstem ukazovat do svého seznamu, aby věděli, kde jsou (trochu podobně jako při skvěání v MergeSortu, který už si uacvřili), a porovnají, zda oba ukazují na jméno stejného kamaráda. Pokud ano, jméno si poznaménají. Pokud ne, tak ten, u kterého je jméno pod prstem v abecedě dříve, posune prst dál.

Teď je vhodný okamžik zhodnotit, zda vše funguje tak, jak si naše dvojka přeje. Nejprve trochu formalizujme ilouhu. Na vstupní dostáváme dvě množiny a vrátit máme jejich průnik, tedy právě ty prvky, které se vyskytnou v obou množinách. Záměrně tím, že nahledneme, že jsme žádný takový prvek nevynechali: postup s ukazovanými prsty způsobuje (stejně jako v MergeSortu), že procházíme prvky obou množin dohromady v seříděném pořadí. Tedy pokud se prvek vyskytoval v obou množinách, neodjedeme z něj v jednom seznamu dokud na něj nenarazíme ve druhém. Posuneme se až tehdy, když je ten druhý prvek v abecedě za námi. Takže každý prvek patříčí do průniku najdeme.

Také se nám nemůže stát, že jsme do průniku zahrnutí prvek, který tam nepatří – přidáme ho jediné tehdy, když byl v obou seznamech, a tedy tam patří.

Ještě nám zbývá zjistit, jak je jejich řešení časové a paměťové náročné, označme si jako N počet lidí v delším seznamu. Na seřídění seznamů bylo potřeba $O(N \cdot \log N)$ času a $O(N)$ paměti. V druhé fázi už Kevin se Sárou seznamy jen prošel v čase $O(N)$, a potřebují místo, kam si poznaménat nadšené kamarády – $O(N)$.

Možná by vás mohlo zarážet, že tu nemluvíme o složitostech výkajících se kratšího seznamu. Jelikož O -čková notace vyjadřuje složitost v nehoršším případě a se seznamy používáme stejné operace, stačí nám uvážovat složitost právě pouze pro ten delší.

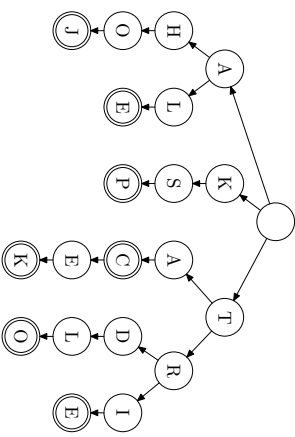
Za takového řešení získáte plný počet bodů.

Program (Python 3):

`http://ksp.mff.cuni.cz/viz/29-21-5.py`

Nikamné jde to i rychlejší, např. když použijeme vhodnou strukturu zvanou *trie*.³ Trie je strom, v jehož každém vrcholu je jeden znak. Slova jsou reprezentována znaky na cestě

z kořene. Tihl pro slova **AMUJ**, **ALE**, **KSP**, **TRIE**, **TRDLO**, **TAC** a **TACEK** si můžete prohlédnout na obrázku. Dvojitým kolečkem značíme, že daným znakem končí slovo.



Jak to tedy celé uděláme? Nejprve si postavíme tři z jednoho ze seznamů, nezáleží ze kterého. Začínáme s prázdným stromem a postupně do něj vkládáme jednotlivá jména.

Teď chceme zjistit, která jména z druhého seznamu byla i v tom prvním. Jednoduché postupně zkusíme každé vyhledat v daném seznamu, a její.

Nezapomente kontrolovat, že jsme opravdu skončili v označeném vrcholu, neodecme mít v seznamu jméno Petr, když mezi kamarády máme pouze Petru. Pokud jméno v tři ne nalzžeme, kamaráda si nepoznaménáme.

Kolik nám to zabere času? Označme si A délku seznamu, ze kterého stavíme tři. B délkou druhého seznamu a jako d délku (libovolného) jména.

Vytvoření trie trvalo nejdéle $O(A \cdot d)$, protože jsme pro každé jméno a každé jeho písmenko museli přidat, projít jeden vrchol. Vyhledání jednoho jména trvá nejvíce $O(d)$, postupně porovnáváme každý znak, a vyhledáváme B jmen, dohromady tedy $O(B \cdot d)$.

Ovšem my ze zadání víme, že d je nejvíce 42, je to tedy (pro tuto ilouhu malá) konstanta a můžeme jí zanedbat. Výsledná časová složitost tak je $O(N)$, kde N je maximum z A a B .

Ve skutečnosti je časová složitost lineární s celkovou velikostí vstupní (součet písmen ve slovech) i bez omezení na délku jmen. To se projevív např. když bychom měli spoustu krátkých slov a jedno hodně dlouhé, kdy bude přibližně rovna součtu délek seznamu a délky dlouhého slova.

Paměťová složitost je $O(A \cdot d)$, druhý seznam dokonce ani nepotřebujeme načítat do paměti celý.

Program (Python 3):

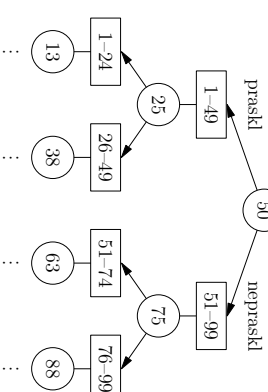
`http://ksp.mff.cuni.cz/viz/29-21-5-trie.py`

Zuzka Šimečková *© Katka Zakratská*

29-21-6 Devadesát devět pater

Pokadže, když Petr shodí balónek z n -tého patra, mohlou nastat dvě situace: Pokud se balónek rozkousí, je jasné, že musíme házet z větší výšky, a tedy už stačí prozkoušet patra $n + 1, n + 2, \dots, 99$. Pokud se ovšem balónek rozhlje, přičkzejí v úvalu již pouze patra $1, 2, \dots, n$. Při hodu z větší vzdálenosti se balónek určité taky rozhlje, ale z větší vzdálenosti by Petr měl zbytečně menší pravděpodobnost, že protivníka trefí.

Protože se snažíme dosáhnout co nejmenšího počtu pokusů i v nehorším případě, chceme si být jisti, že se každým pokusem zbavíme co největšího počtu pater. Máme tedy úsek pater, které ještě přiřadív v úvalu, a shodíme balónek z prostředního patra tohoto úseku. Přesněji řečeno, shodíme balónek z patra $n/2$, pro liché n zaokrouhlíme (umžeme si ověřit, že je jedno na kterou stranu). Všímeleme si, že tím se každým pokusem polovívá pater zbavíme, až nám nakonec zbude pouze jedno jediné. Pro našich 99 pater tedy najdív shodíme z 50. patra. Pokud praskne, shodíme z 25, pokud nepraskne, ze 75. patra. V obou případech nám po těchto dvou pokusech zbývá prověřit už jen 25 pater.



Počet pater, které přiřadív v úvalu, se bude postupně zmenšovat z 99 na 50, 25, 13, 7, 4, 2 a 1 patro. Po nejvíce sedmi pokusech tedy vždycky zjistíme ideální vzdálenost.

Lehčí varianta pro 9 pater funguje identicky, akorát nám na zjištění stačí čtyři pokusy (5, 3, 2, 1).

Pro obecné P určité umíme postupovat stejně, jak ovšem spočítáme celkový počet kroků? Víme, že počet možných pater se bude postupně zmenšovat z P na $\lfloor P/2 \rfloor$, $\lfloor P/4 \rfloor$, $\lfloor P/8 \rfloor, \dots, 1$. Potřebujeme tedy odpovědět na otázku, kolikrát můžeme P vydělit dvěma, abychom došli až k jedničce. Přesně na tuto otázku odpovídá funkce logaritmus, a řečneme, že celkem provedeme $\lceil \log_2 P \rceil$ pokusů.

Tento algoritmus se nazývá binární vyhledávání nebo také metoda plnění interválů. Více se o něm můžete dočíst v naší kuchárce.⁴

Jan Gocník *© Dominik Šimr*

⁴ `http://ksp.mff.cuni.cz/viz/kucharky/vyhledavaci-stromy`